Single Page Application for Project Website Generation using JavaScript

Kyle Hanson: kylehanson485@u.boisestate.edu
Jack Cunningham: jackcunningham@u.boisestate.edu
Logan Ankarberg: loganankarberg@u.boisestate.edu
Patrick Santana: patricksantana@u.boisestate.edu

06/20/2023

Abstract

This project is a web application utilizing HTML, CSS, JS and a couple JS frameworks. These technologies were used to build an application that generates a personal project web page for prospective software engineers. A downloadable file will allow users to personalize their pages how they see fit once the page has been generated. A login system as well as a questionnaire are other features included within the site.

1 Introduction

In this report, we go over our web page generator written in JavaScript including the 'whats?', 'whys?', and details of our implementation. We first give an overview of the language including reasons of motivation to choose JavaScript for our project as well as some details, features, and shortcomings of JavaScript. We will then go over why we chose to implement a web page generator and a broad overview of the stack of our application. From here we'll discuss details of the major features of our application and then wrap it up with our overall impressions and thoughts of working with JavaScript and our finished product.

2 JavaScript Overview

The main motivation of learning JavaScript was that all members of the group were relatively/completely new to JavaScript. So it was a major learning experience for all of us diving into this project. There were a handful of positives and negatives learning JavaScript. Something we enjoyed about JavaScript is how mellow the learning curve was. JavaScript is perfect to learn once a programmer has a solid grasp on HTML + CSS. This is because

when first learning JS, it is usually just messing around with the HTML elements as well as playing with CSS styles. The only difficult part about this is learning the new syntax as well as growing an understanding of how all of these languages tie together to create a web page (which code runs when, etc..) and what exactly a web page is. Learning the syntax was also very mellow. It is a dynamically-typed interpreted language which made it much easier to work with HTML elements and its child and parent nodes due to not assuming the type on initialization. JS is also very versatile; event listeners, functional, and object oriented programming are all supported. One feature that makes JS confusing is the heavy use of anonymous functions and passing functions as parameters. This isn't exclusive to JavaScript, but it is found everywhere in JS code.

Another motivation of learning JavaScript is it is one of the most popular programming languages out there and it does not look like that will change anytime soon. JS practically lives everywhere in modern websites. Typically, it is found on the client-side of a website. Fun fact, 97% of all websites on the internet utilize JavaScript on the front end. But now a days, you can find JS client side, server side, through APIs, and more recently - embedded systems. So, it is a language that is in extreme demand. With the language being so popular, it is well documented and there are plenty of libraries and frameworks out there that best fits the programmer's needs. In our project, we utilized node.js and express.js for the scaffolding and templating of our application. Having plenty of libraries and frameworks with juicy documentation allows the programmer to save hours, days, maybe even weeks of programming. So having these makes JavaScript a powerful language to use. We knew this coming in, so it motivated us to try an idea for our website that wasn't exactly simple and trivial to implement.

2.1 Unique Features of JavaScript

Turning to JavaScript's unique features, it is apparent that JavaScript's basic syntax is heavily inspired by C++ and Java. Knowing one of these languages can drastically improve a programmer's ability to learn JS. However, the inner workings of JavaScript are more similar to dynamically typed, interpreted languages, such as Python or Ruby. As a dynamically typed language, the interpreter assigns variable types at runtime based on the variable's value at the moment it is read, thereby allowing the types of JS variables to evolve over their lifetime. Variables are declared using the keywords var, let, or const. Though let and const are more commonly found in code written after 2015, var is still supported and must be used when running JS code in older browsers.[3]

JS is also unique because it does not force programmers towards one programming paradigm. JS supports the functional programming paradigm which we heavily used in our application. So, programming may be done with an eye towards developing and applying interdependent functions rather than relying so heavily on imperative statements to update the program state. In addition, all non-primitive data types are treated as objects. Therefore, JS supports first class functions, and it is possible to treat functions as values by assigning them to a variable or passing them around as objects. JS also supports higher order functions that can take functions as arguments as well as return them.[2]

Though JS is not a class-based, object-oriented language, object-oriented designs are still supported. JS uses prototypes instead of classes for inheritance. JS is a prototype-based language that uses prototypical objects as a template for providing properties to new objects. As a result, objects have access to the global Object prototype by default, though an object can also be configured to point to other prototypes to achieve "prototype chaining."[1]

JS also supports procedural design patterns and precise control over program execution. A program's functionality may be broken down into a series of functions or subroutines that improve code readability and reusability.

As a garbage-collected language, JS wasn't designed to support the level of memory control provided in languages with manually managed memory such as C. Therefore, JS is more forgiving, but it is also more difficult to optimize, and it can be less secure. This is indicative of the persisting tension between control and ease of use in programming languages.

3 Project Description

To reiterate the basic idea of our project: we implemented a website that will generate a personal project web page, with some additional features. There is a form that the user fills out to supply the data for the generator; this includes sections for personal information, user skills, user credentials, and other additional info. Once a user submits this info, a web page will be generated for them as well as a link to download that generated page. We also chose to use a database which allowed for persistence of data. This means that the user may choose to make a login so if they don't want to download the page, they may come back to our site, log in, and the user will still have their page.

Another feature is our questionnaire. If the user downloads their generated web page, they will be prompted with this questionnaire. The reason we did this instead of doing a typical google questionnaire like other groups is so we could write even more JavaScript.A google survey would have been easy, but we wanted to write JavaScript as much as we could.

A web page generator was a great idea to implement in JavaScript due to it's versatility. It allowed for us to explore the different features of JS and to write JS for many purposes. On all of the pages, we have several event listeners waiting for various events; buttons to be clicked, input to be given, etc.. This allowed for us to write the authentication, form validation, database communication, and rendering all in JavaScript. This made JS fun to work with since it was being used for all sort of things. And like we said earlier, JavaScript was also used to host, run, and template our app.

Another reason why we chose to do a web page generator is that this service a legitimate one - there are plenty out there that people use. We thought it would be nice if our app had some real functionality that users would want for their own reasons. There are plenty of static web page generators out there, and they're great for users that do not want to make their own web page from scratch. It's not exactly meant to give the user a completely

polished web page, but to give users a head start. They can download their page to manipulate, add, and delete whatever they want to fit their needs.

4 Program Description

Our program was created using Node.js and Express JS. We have a public folder that holds all of the images, JS+CSS files, PDFs, and videos. We also had a routes folder with JS files that controlled what is rendered based on the url that is being accessed. This is also where our database queries happen. The last folder is the views folder that held all of the ejs files that would be rendered during the routes. These ejs files are HTML pages that are able to take in variables when rendering which made templating the data much easier. We also had an app.js file that created our database connection and started the server. The last things we had were package folders that node.js generated, node dependencies that were placed into the node modules folder, and an sql script to create the database and tables.

4.1 Environment

Our application utilized a couple JS frameworks as well as a mySQL database to serve our application.

4.1.1 Node.js

Node.js was used to jump start our application by providing a local web server that handles express.js HTTP requests.

4.1.2 ExpressJS

Express.js was used in conjunction with node.js to provide a framework for our web application including routes, templating, and scaffolding.

4.1.3 MySQL

MySQL allowed for data persistence within our website. All users and their data (name, resumes, etc..) are stored in the database. Most of the routes the user hits within our app will cause a database action. For example, when the user submits the main form, a write will occur. When the user goes to their generated page, a read will occur. As of right now, our website only supports read and write operations from the database.

4.2 Login

The login feature for our application provides a very simple user creation and login system. If that user already has created an account, then they may login. If an account has not been created for that user, they may select “create a new account”. Here, the user will enter

a username, email address, password, and confirm the password. If the data is validated within the constraints of the application, the user is returned to the login form. If the user is then authenticated successfully, they will be routed to the main page of the application.

Login.js demonstrates our application of event-handling and client-side validation using JS. It includes JS that allows the user to switch between the login and create account forms, establishes form messages, and contains the functions for validation that are triggered by events occurring in the form. For example, the create account form checks if the username input element contains at least eight characters. If not, an input error message appears in the form when the user triggers the "blur" event by clicking on another input field. When the user triggers the input event by clicking on the username field to enter a valid input, the input error is cleared from the form.

Users.js contains code for our login page and demonstrates our application of encryption, communication with a database, and server-side validation using JS. When the create account form is submitted, a post request is sent to the server and the route corresponding to the createAccount address is invoked in users.js. The password entered is hashed and salted with twelve rounds using the Bcrypt npm package's hasing algorithm. The username, email, and password values entered are stored in the "createdUsers" table contained within the "cs354" SQL database. When the login form is submitted, a post request is sent to the server and the route corresponding to the Auth address is invoked in users.js. The login username and password entered are assigned to variables. The database is queried for the password stored during account creation that is associated with the username entered during login. If the associated password is found, it is compared to the encrypted version of the password value entered during login. This is accomplished through the use of Bcrypt's compare() method. If the passwords match and authentication is successful, the user will be redirected to the main page of the application (index).

4.3 Main Form (index)

The main form was not too difficult to implement. The logic on this page was just some simple form validation and a write to the database. The user would fill out all of the fields including their personal information such as their name, skills, resume, etc.. Once the user submits the form, an event listener will check the form and validate it to our specifications. The only specifications we required is none of the fields should be empty and the user can't have duplicate skills. After these checks pass, a write to the database happens, the user gets redirected, and a read from the database will generate the HTML page with the user data.

4.4 Generated Page

The generated page was pretty difficult to get everything working. The easy part was creating the basic html for the page, but the hard part was filling in the fields that was input from the index form so that it is personalized for each submission of the form. In the end, we ended up using ejs files to solve the problem of filling in the form information.

When the form was submitted from the index, user variables are passed to the generated ejs file when JavaScript renders it. The next difficult part of the generated page was getting the downloads to work. At first we wanted to zip a folder and have the user download it from there. However, the dependency we were using, JSZip, wasn't working inside our project, for unknown reasons. Because of this, we just provide the user with the generated html page and the css file.

4.5 Questionnaire

For our evaluation, we gave users our local machine and let them choose from a library of different images, videos, and pdfs to create a personalized website. We then had them download the files that were generated which routes them to the questionnaire and fill it out. This questionnaire consisted of five questions. Four questions evaluated how well the tool was to use and how likely the user was to recommend it to a friend on a scale of 1 to 5. The last question was open ended and allowed for the user to leave any recommendations for us. When submitted, JS performs some validation to make sure all inputs are filled in. Once submitted, a post will hit our route and a write is made to our answers table in the database.

5 Conclusion

Overall, very positive experience. JavaScript was a great language to learn/grow more knowledge in. We thought that it was a very easy language to pick up even us having zero/minimal knowledge with JavaScript. All of us would use JavaScript again for a project that could use it. We thought that JS was a very powerful language to use and that we were able to make some powerful features really quickly due to a JS framework. So we recommend using JavaScript for any web development if it is appropriate use case because it was very easy to pick up, it very powerful, and that it is one of the most popular and common programming languages out there with luscious documentation.

5.1 Bugs and Issues

- Dependencies do not automatically install
- The password for the database would be changed when someone committed changes
- Database queries had to be made from the routes that were set up via node.js, not client side JS
- When duplicates are entered into the database, an error is thrown and the page won't appear

6 Future Improvements

In terms of completion, we felt pretty satisfied with the end result since we got the major features in place. However, there are a few features that would make the site feel more 'whole'. One major feature that would probably come next if we had time is the ability to update/edit the web page as well as delete the page. If the user made a mistake when creating their page, they would have to fill out the main form all over again. This also causes stale data in the database since that user is not going to use that previous data. Some other minor things that could've been added were more transitions and animations to make the site 'breathe'.

References

[1] Megida Dillion. Object oriented programming in javascript, February 2020. `https://www.freecodecamp.org/news/how-javascript-implements-oop/`, Last accessed on 2022-12-06.

[2] Arora Sukhjinder. Understanding higher-order functions in javascript, October 2018. `https://blog.bitsrc.io/understanding-higher-order-functions-in-javascript-75461803bad`, Last accessed on 2022-12-06.

[3] W3Schools. Javascript variables, 2020. `https://www.w3schools.com/js/js_variables.asp`, Last accessed on 2022-12-06.

7 README

The first step to setting up the environment is download Node.js and visual studio code. After this, clone our repository to your local storage on your computer. Open the directory in vscode, and navigate to the folder of the project: cd .\cs354app\ in the vscode terminal. After this, you will be in the project folder and now is when you install all of the dependencies needed. The first step is to install express.js: npm install express. You also need a dependency to encrypt the passwords: npm install bcrypt. After that, install mysql, and create a database with the password "mysqldatabase". After this, open up generate_tables.sql in db_scripts in our project, copy into mysql workbench (or any db manager), and run the script. Now we can start the server in vscode: npm start localhost:7777. Now go to any browswer and type localhost:7777 into the search bar to use our website.